Генератор Qr-кодов

План:

1. Вступление
2. Описание файлов программы
3. Описание алгоритма
4. Сопутствующие библиотеки
5. Вывод

Для курсовой работы по теме стеганография была выбрана задача реализовать генератор qr-кодов.

Стеганография – способ передачи и хранения информации с учётом сохранения в тайне самого факта такой передачи или хранения.

А, qr-код – монохроматическая картинка, содержащая в себе, закодированный текст, который можно расшифровать с помощью специальных приложений.

Основной целью этого семестра была реализация алгоритма, преобразовывающего какой-либо текст в qr-код. Планировалось так же реализовать графический интерфейс для данной программы, но в силу некоторых обстоятельств, этого достичь не удалось.

В состав программы входят 3 файла:

- dataEncoder.h - заголовочный файл для класса dataEncoder. Этот класс отвечает за полную обработку последовательности как бит, так и байт данных. в качестве входного параметра конструктора выступает строка текста.

- dataEncoderFunctions.cpp - файл, содержащий описание всех методов(функций) реализуемых в классе dataEncoder.

- main.cpp - файл, отвечающий за входные и выходные данные.

**Описание алгоритма:**

Алгоритм по созданию Qr-кода можно разбить на 6 основных шагов:

1) Кодирование данных

2) Добавление служебной информации

3) Разделение информации на блоки

4) Создание байтов коррекции

5) Объединение блоков

6) Размещение информации на Qr-коде

Теперь несколько слов о каждом шаге в отдельности.

*Кодирование данных:*

В данной работе был рассмотрен вариант побайтового кодирования. За реализацию этого шага отвечает метод – void binaryConverter (на вход он получает переводимый текст, ссылку на массив бит (через эту ссылку забираются изменения из функции) и размер текста).

*Добавление служебной информации:*

Для данной работы был выбран фиксированный уровень коррекции qr-кода – M, т.е допустимое кол-во повреждений qr-кода не должно превышать 15%. Этот шаг так же включает в себя:

- определение номера версии; метод, отвечающий за это – int versionNumber, который помимо возвращаемого значения номера версии отвечает за максимальное кол-во бит в последовательности, кол-во блоков и кол-во байтов коррекции

- добавление служебных полей (способ кодирования и кол-во данных), за выполнение этого модуля отвечают методы – void intToBinary, addition1, addition2

- заполнение, на этом шаге так же происходит перевод последовательности бит в последовательность байт (при помощи метода void bitToIntConverter

*Разделение информации на блоки:*

Сначала определяется кол-во байт в каждом блоке:

Кол-во байт последовательности делится на кол-во блоков данных. Если результат деления не целое число, то определяется остаток от деления.

Этот остаток определяет кол-во дополненных блоков (блоков, в которых данных больше, чем в остальных). Дополненными блоками становятся последние блоки. За реализацию разделения последовательности байт на блоки и их заполнение отвечает метод – void blockFiller (на вход получает: последовательность байт, ссылку на массив блоков (через эту ссылку забираются изменения из функции), кол-во блоков, кол-во байт в блоках и кол-во дополненных блоков).

*Создание байтов коррекции:*

Этот один из самых трудоёмких шагов в создании qr-кода включает в себя несколько подпунктов:

- определение кол-ва байтов коррекции

- выбор генерирующего многочлена (за выполнение отвечает метод polynomGenerator)

- создание байтов коррекции (сюда входят методы: int galua (возвращает значения из поля Галуа G(256)), int reverseGalua (обратные значение поля Галуа), void correctionByteGenerator ( на вход получает ссылку на двойной массив байтов коррекции, через которую забираются изменения из функции, массив блоков и генерирующий многочлен)

*Объединение блоков:*

Из каждого блока данных по очереди берётся один байт информации, когда очередь доходит до последнего блока, из него берётся байт и очередь переходит к первому блоку. Так продолжается до тех пор, пока в каждом блоке не кончатся байты. Если в текущем блоке уже нет байт, то он пропускается. Аналогичным образом надо сделать с блоками байтов коррекции. Они берутся в том же порядке, что и соответствующие блоки данных. За выполнение этого модуля отвечает метод: void blocksMerging, получающий на вход ссылку на последовательность байт, через которую забираются изменения из функции, массив блоков и массив байтов коррекции.

*Размещение информации на qr-коде:*

На этом шаге так же можно выделить несколько этапов:

- расположение корректирующих узоров (если они есть), за их поиск отвечает метод void correctionPatternFiller

- поиск размера холста – void sizeOfCanvasFinder

- заполнение холста информацией – метод void qrCodeFiller, этот метод отвечает за полное создание изображение qr-кода, начиная от создания рамки, заканчивая выводом полученного «изображения».

Здесь я описал лишь основные моменты в создании qr-кода и методы, отвечающие за их реализацию. С полным описанием реализованного алгоритма вы можете ознакомиться в файле Algorithm.md.

**Использованные библиотеки:**

Практически все функции, используемые при создании этого приложения, были написаны вручную. Единственное, вместо динамических массивов, использовались вектора.

**Вывод:**

Не смотря на ряд трудностей, возникших в ходе выполнения работы, реализация данной программы была весьма трудным, но интересным заданием. В последствие, после изучение методов ООП (объектно-ориентированного программирования) и qt частности, планируется вернуться к этой курсовой работе и создать для неё графический вариант представления.